RSA

RSA algorithm is based on factoring assumption which states that,

Let genmodulus be a polynomial time algorithm that, outputs (N, p, q) where N = pq, and p, q are n=bit primes except negligible probability. Then, we define an experiment such that:

1) Run GenModulus to obtain (N, p, q)
2) Adversary is given N, and outputs p', q' > 1
3) The output is defined to be 1 if p'.q' = N, 0 otherwise.

We say that factoring is hard for all probabilistic polynomial time adversaries and that there exists a negligible function negl such that:

Pr [Factor (n) = 1] <= negl(n)

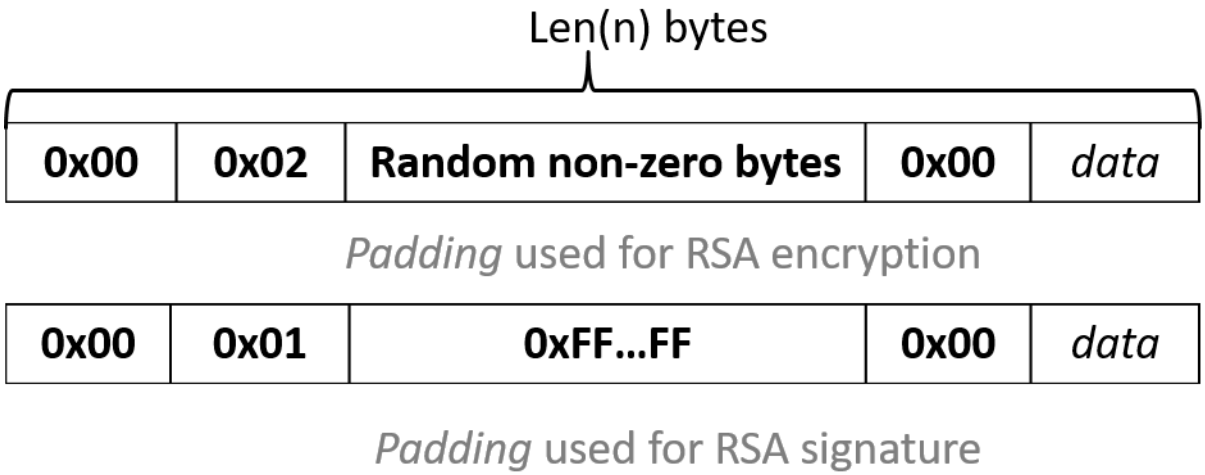
RSA can now be defined as, given N, e, y find x such that x ^e = y mod N.

RSA experiment,

1) Run GenRSA() to obtain (N, e, d)
2) Choose y ← Z*
3) Adversary is given N, e, y and outputs x belonging to Z*
4) Output of the experiment is 1 if x ^e = y mod N. 0 otherwise.

Therefore, we say that RSA problem is hard if for all probabilistic polynomial time algorithms, there exists a negligible function negl such that

Pr[RSA-invA;GenRSA(n) = 1] ≤ negl(n)



Build Algorithm

1) Generate p, q such that they are n-bit prime numbers.
2) Calculate N = p.q and phi_N = (p-1) * (q-1)
3) Calculate public and private kets by choosing e and d, such that e is co-prime with N and phi_N, similarly, d is chosen such that d * e mod phi_N = 1.
4) Define NULL byte, fixed Bytes as well as the bytes to be padded for padded RSA.
5) Message from the user is taken and converted into a list of ascii values per word.
6) The integer list is then encrypted using public key of the receiver.
7) Convert the encrypted list into binary and pad the bytes. We finally have the cipher we can send over a public channel.
8) At receiver's end, padded bytes are first removed.
9) Then the encrypted message list is decrypted and finally converted back to the original format.

Theorem: If the RSA problem is hard relative to GenRSA then the above algorithm has indistinguishable encryptions under chosen-plaintext attacks.